Evaluating formal properties of feature diagram languages

Heymans, P.; Schobbens, P.-Y.; Trigaux, J.-C.; Bontemps, Y.; Matulevicius, Raimundas; Classen, A.

Published in:
IET Software Journal

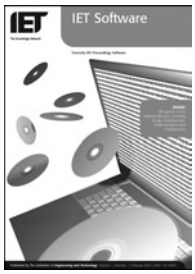
DOI:
10.1049/iet-sen:20070055

Publication date:
2008

Document Version
Early version, also known as pre-print

Published in IET Software
Received on 1st June 2007
Revised on 21st December 2007
doi: 10.1049/iet-sen:20070055

In Special Issue on Language Engineering



ISSN 1751-8806

Evaluating formal properties of feature diagram languages

P. Heymans P.-Y. Schobbens J.-C. Trigaux Y. Bontemps R. Matulevičius A. Classen

Faculty of Computer Science, PReCISE Research Centre, University of Namur, Belgium
E-mail: phe@info.fundp.ac.be

Abstract: Feature diagrams (FDs) are a family of popular modelling languages, mainly used for managing variability in software product lines. FDs were first introduced by Kang *et al.* as part of the feature-oriented domain analysis (FODA) method back in 1990. Since then, various extensions of FODA FDs were devised to compensate for purported ambiguity and lack of precision and expressiveness. Recently, the authors surveyed these notations and provided them with a generic formal syntax and semantics, called free feature diagrams (FFDs). The authors also started investigating the comparative semantics of FFD with respect to other recent formalisations of FD languages. Those results were targeted at improving the quality of FD languages and making the comparison between them more objective.

The previous results are recalled in a self-contained, better illustrated and better motivated fashion. Most importantly, a general method is presented for comparative semantics of FDs grounded in Harel and Rumpe's guidelines for defining formal visual languages and in Krogstie *et al.*'s semiotic quality framework. This method being actually applicable to other visual languages, FDs are also used as a language (re)engineering exemplar throughout the paper.

1 Introduction

A software product line (SPL) is 'a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way' [1]. Software product line engineering (SPLE) is a rapidly emerging software engineering paradigm that institutionalises reuse throughout software development. By adopting SPLE, one expects to benefit from scale economies and thereby improve the cost but also the productivity, time to market and quality of developing software.

One of the main ideas behind SPLE is to dedicate a specific process, named domain engineering, to the development of reusable artefacts, a.k.a. core assets [2]. These core assets are then reused extensively during the development of final products, called application engineering.

Central to the SPLE paradigm is the modelling and management of variability, that is, 'the commonalities and differences in the applications in terms of requirements, architecture, components and test artefacts' [3]. In order to tackle the complexity of variability management, a number of supporting modelling languages have been proposed. To represent variability at the requirements level, an increasingly popular family of notations is the one of feature diagrams (FD). FDs are mostly used to model the variability of application 'features' at a relatively high level of granularity. Their main purposes are: (1) to capture feature commonalities and variabilities; (2) to represent dependencies between features and (3) to determine combinations of features that are allowed and disallowed in the SPL.

During the last 15 years or so, researchers and industries have developed several FD languages. The first and seminal proposal was introduced as part of the feature-oriented

domain analysis (FODA) method back in 1990 [4]. An example of a FODA FD (Inspired from a case study defined in [5]) is given in Fig. 1. It indicates the allowed combinations of features for a family of systems intended to monitor the engine of a car. As is illustrated, FODA features are nodes of a graph, represented by strings and related by various types of edges. On top of the figure, the feature `Monitor Engine System` is called the root (feature) or concept. The nodes can be mandatory or optional. Optional nodes are represented with a hollow circle above their name, for example, `Coolant`. In FODA, mandatory nodes are the ones without a hollow circle. The edges are used to progressively decompose features into more detailed features (also called subfeatures or sometimes sons). FODA offers two kinds of decomposition.

1. and-decomposition, for example, between `Monitor Fuel Consumption` and its sons, `Measures` and `Methods`. It indicates that the latter two features should both be present in all feature combinations where `Monitor Fuel Consumption` is present.

2. xor-decomposition, where edges are linked by an arc, as between `Measures` and its sons, `l/km` and `Gallon/mile`. It indicates that only one of the latter two features should be present in combinations where `Measures` is.

Finally, in FODA, there is also a textual sublanguage called composition rules (or CR) that allows one to express constraints between features that crosscut the tree structure. In Fig. 1, there is only one such constraint, located at the bottom of the diagram. It uses the special keyword 'requires' to indicate that when the former feature is present, the latter should be too.

This paper will later survey our previous contributions [6–8] to give a precise meaning of these and other constructs. However, for now, we retain that the meaning of such diagrams seems to be concerned with the possible combinations (or configurations) of features within any of the products in the SPL – and this was indeed

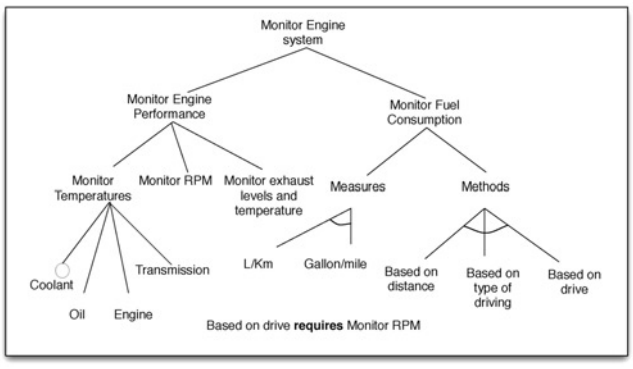


Figure 1 *FODA (OFT) FD: the monitor engine system*

acknowledged in FODA [4] and most of its successors. For instance, one of the allowed feature combinations of the FD in Fig. 1 is: {`Monitor Engine System`, `Monitor Engine Performance`, `Monitor Temperatures`, `Oil`, `Engine`, `Transmission`, `Monitor RPM`, `Monitor exhaust levels and temperature`, `Monitor Fuel Consumption`, `Measures`, `l/km`, `Methods` and `Based on type of driving`}. This combination corresponds to one of the SPL's valid configurations which (1) does not monitor the coolant temperatures, (2) bases the fuel consumption monitoring on the type of driving and (3) uses l/km as a unit of measurement. The FD thus represents an SPL with three variation points (the features `Coolant`, `Measures` and `Methods`) and 12 different valid configurations. The complexity of this particular FD is relatively low. Consequently, the allowed configurations are not so many, and they can be computed relatively easily – provided that we have a precise semantics. However, we should note that there is an exponential factor involved in determining the number of resulting configurations, which mainly depends on the number of variation points and possible choices associated to each of them.

Since Kang *et al.*'s initial proposal [4], several extensions to FODA have been devised as part of the following methods: FORM [9], FeatureRSEB [10] generative programming [11], PLUSS [12], and in the work of the following authors: Riebisch *et al.* [13, 14] and van Gurp *et al.* [15]. A brief overview of these proposals is given in Figs. 2 and 3 where, for each proposed FD language, a quick-facts sheet is given in the left column, whereas an FD based on the same example as Fig. 1 is given in the right column. Note that Figs. 2 and 3 also introduce short names for the languages (e.g. OFT, OFT, RFD...). These will be used in the rest of the paper to facilitate formulations.

When looking at the FDs in Figs. 2 and 3, one immediately sees aesthetic differences among languages. For example, we have observed at least five different notations for the xor-decomposition construct (Fig. 4). These kinds of issues mainly concern concrete syntax, that is, what the users see. Although concrete syntax is an important issue in its own right [16], in this work, we focus on what is really behind the pictures, that is, semantics. We noticed that proponents of FD languages often claimed for added value of their language in terms of precision, unambiguity or expressiveness. Nevertheless, our previous work [6–8, 17] demonstrated that the terminology and evaluation criteria that they used to justify these claims were often vague, and sometimes even misleading.

We think that clearly defined criteria and terminology are paramount to structure the research on effective modelling techniques. Research on FD languages should be no exception. The current status of this research is characterised by a profusion of languages, most of which

doi: 10.1049/iet-sen:20070055

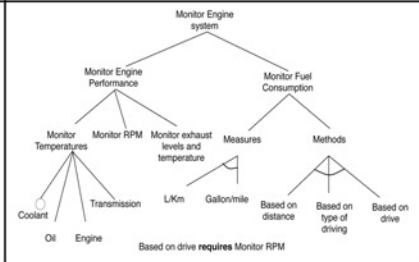
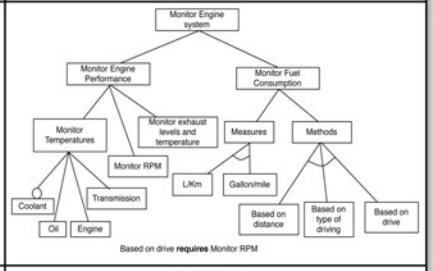
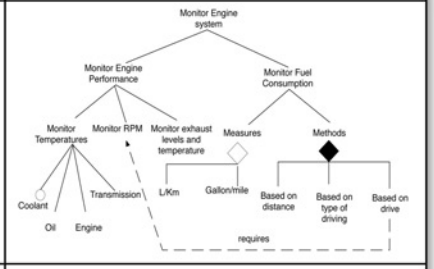
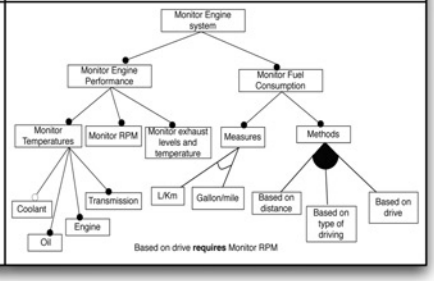
Main Characteristics	Example
Survey Short Name: OFT **Method**: FODA **Author(s)**: [4] **Graph Type**: Tree **Decomposition**: *and, xor, opt* **Constraint Types**: Textual **Formal Semantics**: - **Originally:** none - **A posteriori:** [7]	
Survey Short Name: OFD **Method**: FORM **Author(s)**: [9] **Graph Type**: DAG **Decomposition Types**: *and, xor, opt* **Constraint Types**: Textual **Formal Semantics**: - **Originally:** none - **A posteriori:** [7]	
Survey Short Name: RFD **Method**: FeatuRSEB **Author(s)**: [10] **Graph Type**: DAG **Decomposition Types**: *and, xor, opt, or* **Constraint Types**: Textual & Graphical **Formal Semantics**: - **Originally:** none - **A posteriori:** [7]	
Survey Short Name: GPFT **Method**: Generative Programming **Author(s)**: [11] **Graph Type**: Tree **Decomposition Types**: *and, xor, opt, or* **Constraint Types**: Textual **Formal Semantics**: - **Originally:** none - **A posteriori:** [7,18,21,22,23,24,25]	

Figure 2 *Survey of FD languages (1/2)*

are loosely defined, and loosely compared with their 'competitors'. Although we note that recent work was devoted to the semantic foundations of these languages [18–26], we still lack concrete means to discriminate between these proposals.

This is what the current paper has to offer: a rigorous method to evaluate and compare FD languages, focused on the study of their semantics. This method relies on formally defined criteria and terminology, based on the highest standards in engineering formal languages [27, 28] and situated within a global language quality framework [29, 30].

A minor contribution of this paper is to recall our previous investigations [7, 8, 17] in a self-contained, better illustrated and better motivated way, thereby giving a clearer picture of our overall language (re-)engineering endeavour. Moreover, whereas [7, 17] are more technical papers devoted to formally proving comparison results, the major contribution of this paper is methodological: it proposes a comprehensive and rigorous method for comparative semantics. The method is applied to FD, but it is actually more general and applies to all other visual languages. Therefore the present paper can be regarded as a language engineering exercise that might serve as an exemplar for application to other languages. An abstract of this paper appeared in [31].

This paper is structured as follows. In Section 2, we situate our work within SEQUAL [29, 30], a comprehensive framework for assessing the quality of modelling languages. Section 3, based on [27, 28] recalls the basic concepts on which our method relies: concrete syntax, abstract syntax, semantic function and semantic domain. An extensive illustration of these concepts on the FD language OFD [9] ends this section. The general method for comparative semantics is described in Section 4. It emphasises how

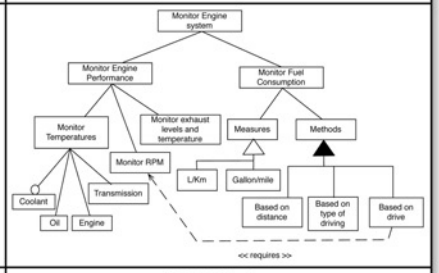
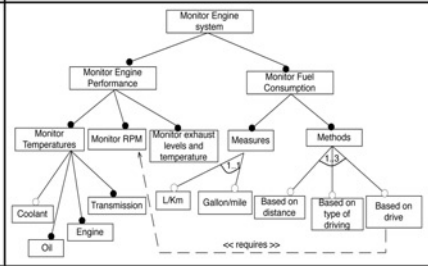
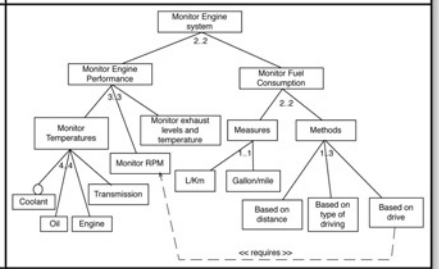
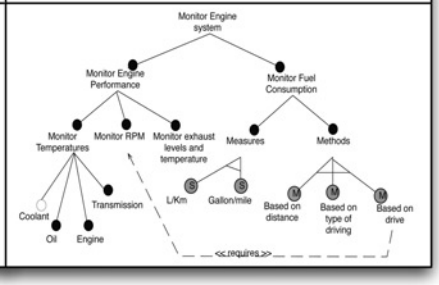
Main Characteristics	Example
Survey Short Name: VBFD **Method**: / **Author(s)**: [15] **Graph Type**: DAG **Decomposition Types:** *and, xor, or, opt* **Constraint Types**: Textual & Graphical **Formal Semantics**: - **Originally:** none - **A posteriori:** [7]	
Survey Short Name: EFD **Method**: FORE **Author(s)**: [13] **Graph Type**: DAG **Decomposition Types:** *card, opt* **Constraint Types**: Textual & Graphical **Formal Semantics**: - **Originally:** none - **A posteriori:** [7]	
Survey Short Name: VFD **Method**: / **Author(s)**: [6] **Graph Type**: DAG **Decomposition Types:** *card* **Constraint Types**: / **Formal Semantics**: - **Originally:** [6] - **A posteriori:** [7]	
Survey Short Name: PFT **Method**: PLUSS **Author(s)**: [12] **Graph Type**: Tree **Decomposition Types:** *and, xor, opt, or* **Constraint Types**: Graphical **Formal Semantics**: - **Originally:** none - **A posteriori:** [7]	

Figure 3 *Survey of FD languages (2/2)*

languages with no clearly defined semantics, or with different semantic domains, can be made suitable for comparison. It also defines formal criteria to compare the languages once they have been made comparable. Section 5 summarises the results obtained so far [7, 8, 17] by applying the method. The paper finishes by discussing the current limitations of the method (Section 6), the research challenges that are still ahead (Section 7), and the conclusions (Section 8).

2 Quality in modelling

2.1 Model quality

Assessing and improving the quality of models and languages is a complex and multidimensional task. A comprehensive view of the concerns involved is given in the SEQUAL (semiotic quality) framework, developed over the last decade by Krogstie *et al.* [30] as an extension of Lindland *et al.*'s original framework [32]. SEQUAL considers models as signs and, based on this, distinguishes 'semiotic levels' in the act of communicating through models: physical, empirical, syntactic, semantic, pragmatic and social. SEQUAL advocates that for an effective use of modelling, quality must be pursued at all the levels of the 'semiotic ladder'. SEQUAL adheres to a constructivistic world-view that recognises model creation as part of a dialog between participants (that is those involved in the creation and usage of models) whose knowledge changes as the process takes place.

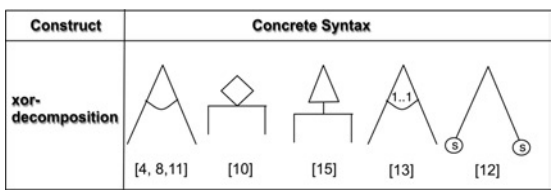


Figure 4 *Concrete syntaxes for xor-decomposition*

doi: 10.1049/iet-sen:20070055

An accurate description of SEQUAL as well as means to pursue and measure the achievement of the quality goals (physical quality, empirical quality...) can be found in [29, 30]. By the extent of the envisaged model qualities and its neutrality w.r.t. a particular kind of models, SEQUAL is arguably the most complete framework we know of. However, it does not detail how to carry out specific quality evaluation or improvement tasks, unlike other proposals, for example, [33, 34] or [35]. Nevertheless, SEQUAL is amenable to specific criteria and guidelines by tailoring. Its main advantages are that (1) it helps situate one's investigations within a comprehensive quality space, (2) it acts as a checklist of qualities to be pursued and (3) it recommends general guidelines on how to proceed.

Our investigation is targeted at a specific kind of models, namely, FDs. Furthermore, we mainly target semantic and pragmatic qualities of these models, which we have found to be somehow neglected in the current state of the art [7, 8]. So doing, we will see that we inevitably interfere with other qualities, mainly syntactic quality.

The problem with evaluating model quality is that the representative objects of study – that is, models – do not always exist, or at least are not easily available. And this is indeed the case for FDs, which (1) are an emerging modelling paradigm and (2) have the purpose of representing highly strategic company information. Therefore representative models are almost nowhere to find, except illustrative examples in research papers [36]. At this stage, we thus thought that we should concentrate on improving the quality of FD languages before any standardisation is attempted and they hopefully become widespread in industry.

2.2 Language quality

In [29], SEQUAL has been adapted to evaluate 'language appropriateness' (Fig. 5). Six quality areas were proposed. Domain appropriateness means that language L must be powerful enough to express anything in the domain D, and that, on the other hand, it should not be possible to express things that are not in D. Participant language knowledge appropriateness measures how the statements of L used by the participants match the explicit knowledge K of the participants. Knowledge externalisability appropriateness means that there are no statements in K that cannot be expressed in L. Comprehensibility appropriateness means that language users understand all possible statements of L. Technical actor interpretation appropriateness defines the degree to which the language lends itself to automatic reasoning and supports analysability and executability. Finally, organisational appropriateness relates L to standards and other needs within the organisational context of modelling.

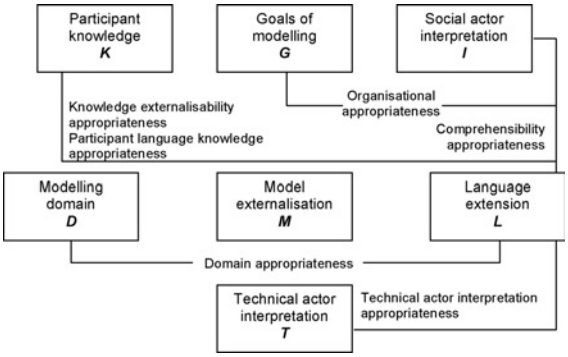


Figure 5 *SEQUAL: language quality (adapted from [29, 30])*

Not being able to assess model qualities directly, our investigations were targeted at three main language qualities: domain appropriateness, comprehensibility appropriateness and technical actor interpretation appropriateness. In particular, we studied four criteria: expressiveness, embeddability, succinctness and complexity. The rigourous definition of those criteria as well as a discussion of how they match our target language qualities are provided in Section 4.

3 Engineering formal modelling languages

In their illuminating papers, "Meaningful modelling: What's the semantics of 'semantics'?" [27] and 'Syntax, semantics and all that stuff. Part I: the basic stuff' [28], Harel and Rumpe recognise that: 'Much confusion surrounds the proper definition of complex modelling languages [...]. At the root of the problem is insufficient regard for the crucial distinction between syntax and true semantics and a failure to adhere to the nature and the purpose of each' [27].

Although they are far less complex than, for example, the unified modelling language (UML) [37], we demonstrated in previous papers [6, 8] that FDs are also 'victims' of similar 'mistreatments'. In [27, 28], one of Harel and Rumpe's main motivations is to suggest how to improve the UML. The objective of this section is to recall the notions of (formal) syntax and semantics from [27, 28]. In the subsequent sections, we will show how, based on these notions, we have devised an approach to (re)define, assess and compare languages.

3.1 Basic notions

Very basically, the term 'syntax' refers to the notation that a language offers to its users. 'Semantics', on the other hand, refers to the meanings that its expressions (programs, diagrams, sentences,...) are aimed to convey.

For Harel and Rumpe, the unambiguous definition of a modelling language, be it textual or graphical, must consist of three equally necessary elements: a syntactic domain ($\mathcal{L}$), a semantic domain ($\mathcal{S}$) and a semantic function ($\mathcal{M}$). Furthermore, to keep the risk of ambiguity at its minimum,

$\mathcal{L}$, $\mathcal{S}$ and $\mathcal{M}$ must all be defined formally, that is, mathematically. A language with such formal $\mathcal{L}$, $\mathcal{S}$ and $\mathcal{M}$ is called a formal language.

In the context of FDs, it might be extremely useful to have a tool that tells whether a given diagram allows for at least one feature configuration, or if it is overconstrained and thereby allows none. For a real-size FD, this verification is far from trivial, and can be very time-consuming and error-prone if left to humans. This type of verification is known as satisfiability checking and is one of the many FD-related tasks that can be automated [7, 8] (see also Section 4.3). If we want to engineer languages in such a way that no ambiguity is left to tool developers, then we need to make them formal. Only then can we prove the correctness, and study the efficiency (computational complexity), of its supporting algorithms. And only then can we study the formal language properties such as expressiveness, succinctness and embeddability (Section 4.3).

Formal semantics is also useful if a language is simply used as a means of communication between human participants [16]: one can refer to it in case of doubt, thereby avoiding unintended meanings.

The above may seem all too obvious to some readers. However, during our survey, we could observe that many FD languages were never formally defined, despite the actual simplicity of the task (as we will see in Section 3.4). A tentative explanation to this situation can be found in [27, 28] where a set of frequent misconceptions about formal semantics are enumerated, for example, 'Semantics is the metamodel', 'Semantics is dealing with behaviour', 'Semantics is being executable', 'Semantics means looking mathematical' etc. But, for now, we return to the definitions of $\mathcal{L}$, $\mathcal{S}$ and $\mathcal{M}$, which we make more precise.

3.2 Syntax

In diagrammatic (a.k.a. visual or graphical) languages, such as FDs, basic expressions include lines, arrows, closed curves, boxes and composition mechanisms involve connectivity, partitioning and 'insideness' [27]. These form the physical representation of the data (on screen or on paper) which is known as concrete syntax.

For visual languages, it appears particularly difficult to define rigid syntactic rules that clearly segregate between valid and forbidden diagrams (this is also true, to a lesser extent, of textual languages). Furthermore, when based on the concrete syntax, the expression of the semantic interpretation rules is polluted by considerations related to visualisation. This is why the common practice in the aforementioned areas is to define the semantics of a language based on a so-called abstract syntax. Nevertheless, most of the (informal) definitions of the semantics of FDs we found in the literature were based on concrete syntax, usually discussed on FD examples and incomplete.

The abstract syntax ($\mathcal{L}$) is a representation of data that is independent of its physical representation and of the machine-internal structures and encodings. The set of all data that comply with a given abstract syntax is called the syntactic domain.

For visual languages, the two most widespread ways to define an abstract syntax are: (1) mathematical notation (set theory) and (2) meta-modelling. In the latter case, the abstract syntax is described by a so-called meta-model describing what is a well-formed (allowed) diagram. A meta-model is usually a UML class diagram, possibly complemented with object constraint language (OCL) rules [37]. This format has the main advantage to be easily readable and to facilitate some tool implementation tasks, especially persistent storage of diagrams in a repository. Nevertheless, we prefer the mathematical format for its greater universality, unambiguity, conciseness and suitability to undergo rigorous proofs [7]. As an example, in Section 3.4, we define $\mathcal{L}_{\mathrm{OFD}}$, the abstract syntax of OFD. In Section 4.2, we will recall how we managed to provide an abstract syntax for several FD languages at once through a generic mathematical structure that we called free feature diagram (FFD) [7, 8].

3.3 Semantics

Once we have a rigid set of syntactic rules, the role of a semantics is to assign an unambiguous meaning to each syntactically correct diagram. Harel and Rumpe recognise that '[a]greement on a language's meaning is partly a sociological process, without which the communicated data are worthless' [27]. As we have seen in Section 2, this point of view is acknowledged and further elaborated in [29, 30]. The sociological aspects of semantics are, however, out of the scope of the current investigation, although some will be discussed in Section 7. We follow [27, 28] and consider a semantics to have two main constituents: a semantic domain ($\mathcal{S}$) and a semantic function ($\mathcal{M}$), both to be defined mathematically.

According to [27], the semantic domain '[. . .] specifies the very concepts that exist in the universe of discourse. As such, it serves as an abstraction of reality, capturing decisions about the kinds of things the language should express'. Typically, $\mathcal{S}$ is a mathematical domain built to have the same structure as the real-world objects the language is used to account for, up to some level of fidelity. The semantic domain that we have proposed for OFD and for the other surveyed FD languages is named product lines (PL) [7, 8]. It is recalled in Section 3.4. Looking at the semantic domain is necessary to compare two semantic definitions, as we will show in Section 4.

The second constituent of a semantics is $\mathcal{M}$, the semantic function. $\mathcal{M}$ relates $\mathcal{L}$ and $\mathcal{S}$ by assigning a meaning to each syntactically valid diagram. Its signature is thus $\mathcal{M} : \mathcal{L} \rightarrow \mathcal{S}$. In the case of OFD, for example,

doi: 10.1049/iet-sen:20070055

we have the signature $\mathcal{M}_{\text{OFD}} : \mathcal{L}_{\text{OFD}} \to \text{PL}$. The definition of $\mathcal{M}_{\text{OFD}}$ is given in Section 3.4. Semantic functions, like all mathematical functions, can be described in a number of ways. In our example, we found it convenient to express it as four fairly simple rules. These rules describe declaratively how the syntactic constructs used in the diagram constrain its associated object in PL. The generic semantic function of FFD ($\mathcal{M}_{\text{FFD}}$) is not much more complex, as it counts just one additional simple rule (Section 4.2).

Being a function, $\mathcal{M}$ overrules ambiguity. In the previous work [6], we could reconstruct a semantic function from the original plain English definition of OFT [4]. We could thereby dismiss criticisms of ambiguity [14].

In addition, a total semantic function ensures that the semantics is complete. In Section 4.3, we will address the converse question: is every element in $\mathcal{S}$ expressible by a diagram in $\mathcal{L}$? This will help us judge the expressiveness of a language, another term confusingly used in the literature.

3.4 Example: FORM feature diagrams (OFD)

We use OFD [9] (the first extension of OFT [4]) as an illustrative example of the language definition principles that we just exposed. It is also used as a prototypical example of the FD languages that we have generalised in our FFD construct (Section 4.2).

OFD is used very frequently in the literature. There are two variants of it. The simplest just extends OFT with the possibility to draw directed acyclic graphs (DAGs) instead of being limited to trees. The second is more complex as it further adds three completely new constructs in FDs: layers, implementation relationships and generalisation/specialisation relationships. Here, we only look at the simpler form. The more complex form is briefly discussed in Section 7.

The syntactic domain of OFD ($\mathcal{L}_{\text{OFD}}$): From the point of view of concrete syntax, OFD are graphical combinations of elementary symbols such as boxes (features), strings (feature names and textual constraints), lines (feature decomposition) and circles (on top of optional features). Defining the allowed combinations of these symbols involves describing where they should be placed, which size and colour they should have etc. This is not necessarily difficult, but bulky. The abstract syntax defines the allowed essential syntactic structures behind these visualisation details.

A natural mathematical abstract syntax for OFD is given in Definition 1 and illustrated in Fig. 6. We observe that an OFD is essentially a graph, that is, a set of nodes (let us call it N) and a relation between nodes (DE $\subseteq N \times N$). Features naturally map to elements of N, and

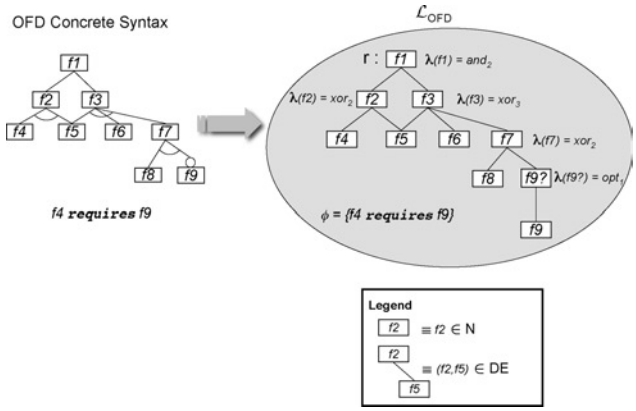


Figure 6 *OFD's abstract syntax: $\mathcal{L}_{OFD}$*

decomposition edges to elements of DE. In an OFD, there is always a unique root feature. We will call it r and thus have $r \in N$. If a feature f_1 is decomposed into features f_2 and f_3, then we have $(f_1, f_2) \in \text{DE}$ and $(f_1, f_3) \in \text{DE}$. Although decomposition is represented in the concrete syntax by plain lines instead of arrows, it does have a direction. In OFD (as in other FD languages), the direction is represented by a feature being placed graphically above its subfeatures. Also, in OFD, decomposition must not have cycles (which is less restrictive than having to be a tree, like in OFT). Hence, DE must be a DAG.

To abstract the various kinds of decompositions, we introduce a labelling function λ that maps each node to a boolean and_s or xor_s operator, respectively, for and- and xor-decomposition. s denotes the arity of the operator and must be equal to the number of subnodes (subfeatures) of the labelled node. The signature of λ is thus $\lambda : N \to \text{NT}$, and NT (node type) is a set of boolean operators. It contains operators and_1, and_2, and_3, ... as well as operators xor_2, xor_3, ... In addition, NT also contains and_0 and opt_1, explained further in the section. So, if f_1 is and-decomposed into f_2 and f_3, we will have $\lambda(f_1) = \text{and}_2$. A node like f_1 will thus be called an and_2-node, and sometimes simply and-node. We adopt similar terminological conventions for nodes labelled with other (types of) operators. Another convention is that terminal features, that is features which have no subfeature, are λ-labelled with and_0.

The abstract syntax of optional features (those with a hollow circle on top in the concrete syntax) is a little trickier. For each such feature with label `f1` in the concrete diagram, the abstract diagram possesses two nodes, say f_1 and $f_1?$. $f_1?$ is introduced as an intermediate node between f_1 and those nodes which should have been its supernodes, had it not been optional. The only (direct) subnode of $f_1?$ is f_1 and $\lambda(f_1?) = \text{opt}_1$. opt_1 is the boolean operator that always returns TRUE. This way to define the abstract syntax of optional nodes came after noticing that they actually played a role similar to the and- and xor-decomposition,

except for the fact that this kind of operator only acts upon one subnode.

An important concept we introduce in $\mathcal{L}_{\text{OFD}}$ and which we also need in the generalised abstract syntax $\mathcal{L}_{\text{FFD}}$ (Section 4.2) is the concept of primitive node (also called primitive feature; see Definition 1). As also recognised by other authors [22, 38], there is currently no agreement on the following question: are all features equally relevant to define the set of possible products that the FD stands for? This question primarily addresses semantics, but has consequences for the syntax.

For example, in Fig. 1, one could question whether the (absence or presence of the) feature `Measures` is useful to describe a product, or if (the absence or presence of) its subfeatures, `l/km` and `Gallon/mile`, suffice(s). Since there was no agreement in the literature, we adopted a neutral formalisation. Our solution accounts for the fact that the modeller can consider only part of the features as relevant. Although there is no construct in the concrete syntax (neither for OFD nor any other FD language we know), we need to introduce one in the abstract syntax, namely a subset P of N ($P \subseteq N$). We will see the impact of P when we address the semantics of OFD. Finally, although we leave it to the modeller to determine P, we reasonably expect that P (1) contains terminal features and (2) does not contain opt-nodes. But we need not impose these rules.

The last part of OFD's concrete notation that $\mathcal{L}_{\text{OFD}}$ should account for is the textual constraint language, called 'composition rules' (CR). In the concrete diagram, the language is used to specify a (possibly empty) set of rules, usually located at the bottom of the diagram. In the abstract syntax, we call the set of rules Φ and define it as a set of words obeying the following production rule: CR ::= f_1 (`requires`|`mutex`)f_2, where $f_1, f_2 \in P$.

Definition 1 (Original Feature Diagram): An OFD $d \in \mathcal{L}_{\text{OFD}}$ is a tuple $(N, r, \lambda, DE, \Phi)$ where:

1. N is its set of nodes;

2. $P \subseteq N$ is its set of primitive nodes;

3. $r \in N$ is the root;

4. $\lambda : N \to \text{NT}$ labels each node with an operator from NT, where $\text{NT} = \text{and} \cup \text{xor} \cup \{\text{opt}_1\}$, that is, a set of boolean functions (operators) where:

- and is the set of operators and_s ($s \in \mathbb{N}$), that return `TRUE` iff all their s arguments are `TRUE`;

- xor is the set of operators xor_s ($s \in \mathbb{N} \setminus \{0, 1\}$) that return `TRUE` iff exactly one of their s arguments is `TRUE`;

- opt_1 is the operator that returns `TRUE`

The semantics of those operators is recalled here for convenience only. By definition, it is not a part of OFD's abstract syntax.

5. $\text{DE} \subseteq N \times N$ is the set of decomposition edges; $(n, n') \in \text{DE}$, alternatively noted $n \to n'$;

6. $\Phi \subseteq \text{CR}$ are the textual constraints, when CR ::= f_1(`requires`|`mutex`)f_2 and $f_1, f_2 \in P$.

Furthermore, d must satisfy the following well-formedness rules.

1. Only r has no parent: $\forall n \in N \cdot (\nexists n' \in N \cdot n' \to n) \Leftrightarrow n = r$.

2. DE is acyclic: $\nexists n_1, \ldots, n_k \in N \cdot n_1 \to \cdots \to n_k \to n_1$.

3. Node operators are of adequate arities: $\forall n \in N \cdot \lambda(n) = \text{op}_k \wedge k = \sharp\{(n, n') | n \to n'\}$.

4. Terminal features are and_0-labelled: $\forall n \in N \cdot (\nexists n' \in N \cdot n \to n') \Leftrightarrow \lambda(n) = \text{and}_0$.

Note that the first two well-formedness rules above should be enforced at the level of the concrete syntax (for example, by a graphical OFD editor), whereas the last two rules should be guaranteed when moving from the concrete to the abstract syntax, and the modeller should not care about them.

The semantic domain of OFD ($\mathcal{S}_{OFD} = PL$). In the surveyed literature, there seems to be an agreement that FDs are meant to represent the sets of products, and each product is seen as a combination of features. These tenets were present from the beginning in OFT [4] and were adopted without much controversy in its extensions, including OFD. In particular, none of the surveyed languages attempted to further define the 'contents' of a feature beyond its name (viz., the labels appearing in the nodes of the FD), except for some recent work [23, 39] (Section 7.2). We published the first formalisation of a semantic domain specifically devoted to FDs in [6]. In this semantic domain, named PL, the atomic building blocks are features (nodes), a bit in the same way that propositions are the atomic building blocks in the semantic domain of propositional logic [40]. However, we want to leave the flexibility to the modellers to decide which features are relevant for them to discriminate products, so we use P instead of N. Definition 2 formalises the notions of product and product line, relying on the more general notion of configuration.

Definition 2 (Configuration, Product, Product Line):
- A configuration is a set of features/nodes, that is, any element of $\mathcal{P}N$.

- A product is a configuration that contains only primitive features, that is, any element of $\mathcal{P}P$.

doi: 10.1049/iet-sen:20070055

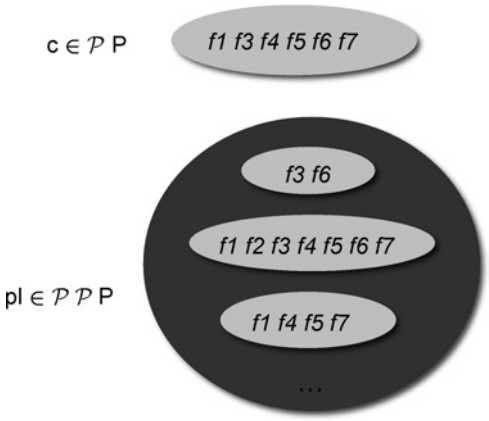


Figure 7 *OFD's semantic domain: $\mathcal{L}_{OFD} = PL$*

- A product line is a set of products, that is, any element of $PL = \mathcal{PPP}$.

Fig. 7 gives an illustration of this. Like a configuration, a product, say c, is a combination (that is a set) of features (nodes). In this case, c is the set $\{f_1, f_3, f_4, f_5, f_6, f_7\}$. A product line, for example pl, is a set of products. Here, pl is a set of three products: $\{\{f_3, f_6\}, \{f_1, f_2, f_3, f_4, f_5, f_6, f_7\}, \{f_1, f_4, f_5, f_7\}\}$.

Recently, in other formalisation proposals, some authors [18–24, 6] have chosen semantic domains different from PL, for example, using lists instead of sets [18]. How to compare PL with other semantic domains will be discussed in Section 4 but the cause usually turns out to be an implementation bias. For the time being, we complete the definition of the semantics with the semantic function.

The semantic function of OFD ($\mathcal{M}_{OFD} : OFD \rightarrow PL$). In Fig. 8, we depict $\mathcal{M}_{\text{OFD}}$, the semantic function of OFD. To every diagram d, it assigns a PL, noted $\mathcal{M}_{\text{OFD}}[\![d]\!]$. $\mathcal{M}_{\text{OFD}}[\![d]\!]$ is more formally described in Definitions 3 and 4. Definition 3 indicates which set of products is returned by $\mathcal{M}_{\text{OFD}}[\![d]\!]$: the set of the configurations (combinations of features) which are valid w.r.t. d, restricted to their primitive features.

Definition 3 (Semantic function): The semantics of an OFD d is a PL (Definition 2) consisting of the products of d, that is, its valid configurations (Definition 4) restricted to primitive features/nodes: $\mathcal{M}_{\text{OFD}}[\![d]\!] = [\![d]\!] = \{c' | c \vDash d \wedge c' = c \bigcap P\}$.

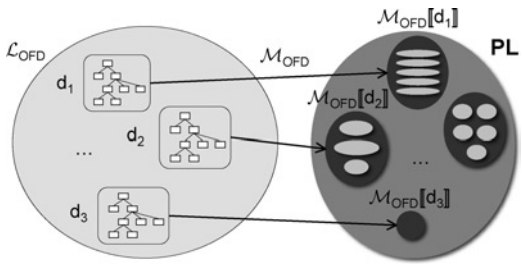


Figure 8 *OFD's semantic function: $\mathcal{M}_{OFD}$*

Definition 4 provides four clear and compact rules telling what in OFD is a valid configuration w.r.t. d. The fact that a configuration c is valid w.r.t. d is noted $c \vDash d$.

Definition 4 (Valid configuration): A configuration $c \in \mathcal{P}N$ is valid for a $d \in \mathcal{L}_{\text{OFD}}$, noted $c \vDash d$, iff:

1. The root is in: $r \in c$.

2. The meaning of nodes is satisfied: If a node $n \in c$, and n has sons $s_1, \ldots, s_k$ and $\lambda(n) = \text{op}_k$, then $\text{op}_k(s_1 \in c, \ldots, s_k \in c)$ must evaluate to TRUE.

3. The configuration must satisfy all textual constraints: $\forall \phi \in \Phi, m \vDash \phi$, where $m \vDash \phi$ means that we replace each node name n in ϕ by the truth value of $n \in c$, evaluate ϕ and obtain TRUE. For instance:

- if ϕ is a CR constraint of the form f_1 `requires` f_2, we say that $m \vDash \phi$ when $(f_1 \in c) \Rightarrow (f_2 \in c)$ evaluates to TRUE;

- if ϕ is a CR constraint of the form f_1 `mutex` f_2, we say that $m \vDash \phi$ when $\neg$ and$_2(f_1 \in c, f_2 \in c)$ evaluates to TRUE.

4. If s is in the configuration and s is not the root, one of its parents n, called its justification, must be too: $\forall s \in N \cdot s \in c \wedge s \neq r \cdot \exists n \in N \cdot n \in c \wedge n \rightarrow s$.

When $\mathcal{M}_{\text{OFD}}[\![d]\!]$ returns an empty set of products, that is the empty PL, it means that d is non-satisfiable (or inconsistent). In Fig. 8, this is the case for d_3. This happens when there is no product combination that can satisfy the constraints in d. Checking consistency, as well as many other tasks, can usually not be performed efficiently just by processing syntax, nor by letting the modeller inspect the diagram. Hence, the utility of defining the semantics in a way that enables faithful implementation into a computer programme automating time-consuming and error-prone tasks.

We now take a closer look at the product validity rules of Definition 4. The application of the rules is illustrated in Fig. 9. We assume that all features in the OFD are primitive, except for f_9? which was generated to account for an optional feature in the concrete syntax.

1. The first rule imposes that the root $(r = f_1)$ appears in every valid product. Hence, the product $\{f_2, f_3, f_4, f_7, f_9\}$, for instance, cannot be part of it.

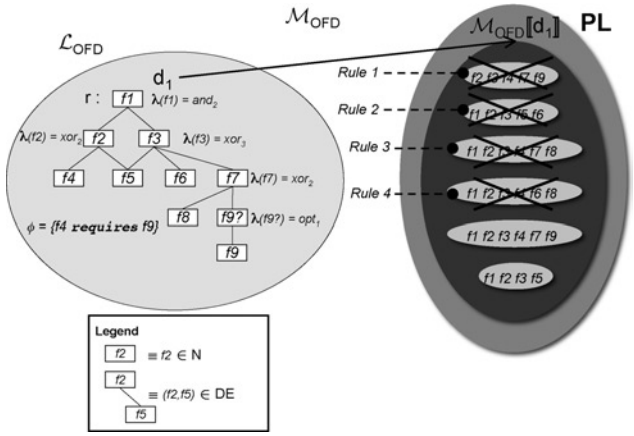


Figure 9 *OFD's semantics: an example*

2. The second rule describes the semantics of the boolean operators coming from the decomposition links and from the optional features. This rule relies on the semantics of the boolean operator opt_1 as well as the operators in and and xor. Their semantics has been recalled earlier in this section. In the example, the rule is applied to discard the product $\{f_1, f_2, f_3, f_5, f_6\}$, say c, because f_3 appears in it together with more than one node among f_5, f_6 and f_7. Indeed, f_3 is labelled with xor_3 and has three sons: f_5, f_6 and f_7. In c, $\text{xor}_3(f_5 \in c, f_6 \in c, f_7 \in c)$ would then evaluate to FALSE.

3. The third rule is similar in spirit to the former, except that it deals with the operators (`requires` and `mutex`) appearing in Φ, the CR constraints accompanying the graphical part of the OFD. When applied to the example, the rule interprets the CR f_4 `requires` f_9 by checking the truth value of $(f_4 \in c) \Rightarrow (f_9 \in c)$, which in the case of the product $c = \{f_1, f_2, f_3, f_4, f_7, f_8\}$ evaluates to FALSE.

4. The fourth and last rule is called the justification rule. It guarantees that, except for the root, a node cannot be present in a valid product without one of its parent nodes being present as well. It says 'one of its parents' because OFDs are DAGs and a node can therefore have multiple parents. In the example, this rule discards the product $c = \{f_1, f_2, f_3, f_4, f_6, f_8\}$ because f_8 belongs to c but f_7, its only parent, does not. The justification rule has been often overlooked in the literature. For example, in [25], a formal semantics of FD is proposed without such a rule. This leads to strongly counter-intuitive semantics. Without the justification rule, the OFD in Fig. 9 would accept, for example, products $\{f_1, f_2, f_3, f_9\}$ or $\{f_1, f_2, f_3, f_4, f_6, f_8\}$ as part of its semantics. Justifications also explain the difference between decomposition through and-nodes and `requires` constraints: the presence of a subfeature is justified by its and-parent, while `requires` gives no justification.

Eventually, when all the rules in Definition 4 have been taken into account and when all the non-primitive features in the products have been removed according to Definition 3, we see that the semantics of the OFD in Fig. 9 comes unambiguously as the following PL, made of two valid products: $\{\{f_1, f_2, f_3, f_4, f_7, f_9\}, \{f_1, f_2, f_3, f_5\}\}$.

4 Comparing FD languages

When languages receive a proper definition of $\mathcal{L}$, $\mathcal{S}$ and $\mathcal{M}$, they can be assessed by means of rigorously defined semantic criteria. Three commonly used criteria are:

- expressiveness: what can the language express?

- embeddability (also called naturalness or macro-eliminability): when translating a diagram to another language, can we keep its structure?

- succinctness: how big are the expressions of a same semantic object?

In addition, in a formal language, one can precisely define decision problems, that is, tasks to be automated. For example, in languages like FDs that have a set as semantic domain, we can state precisely the satisfiability problem: given a diagram d, is $\mathcal{M}[\![d]\!] \neq \emptyset$ true? As we will see, there are many other such questions. Once these problems are formalised, one can ask (1) whether answers (that is computable functions) exist at all to solve this problem (decidability) and (2), if so, what is their relative computational difficulty (complexity).

With respect to the SEQUAL framework (recalled in Section 2), we focus on the following qualities.

- Domain appropriateness is addressed by looking at language expressiveness.

- Comprehensibility appropriateness is addressed by looking at embeddability and succinctness.

- Technical actor interpretation appropriateness is addressed by looking at decidability, complexity and also embeddability and succinctness.

More rigorous definitions of those criteria will be given further in Section 4.3. But first, we need to address the practical concern that not all FD languages may be directly comparable with each other according to those criteria.

Let us assume that we have to compare FD languages $X_1, \ldots, X_n$. We need to have formally defined languages, that is, for language X_i, we need to know $\mathcal{L}_{X_i}$, $\mathcal{S}_{X_i}$ and $\mathcal{M}_{X_i}$. Furthermore, if we want to be able to compare expressiveness, embeddability and succinctness, we also need to have $\mathcal{S}_{X_1} = \mathcal{S}_{X_2} = \cdots = \mathcal{S}_{X_n}$.

However, this ideal situation almost never occurs in practice. Most of the time, we have to cope with:

doi: 10.1049/iet-sen:20070055

• languages that have no formal semantics at all (this is the most frequent case, which we addressed in [7, 8]);

• languages with a formal semantics but defined in quite a different way from what is advocated in Section 3;

• languages with a formal semantics compliant with the recommendations of Section 3, but using a different semantic domain.

We have also started to address instances of the latter two cases recently in [17].

The overall process of comparing FD languages should thus be performed in two steps: (1) make the languages suitable for comparison and (2) make the comparisons.

In Section 4.1, we offer a systematic approach to cope with the first step, in all different situations. In Section 4.2, we recall FFD, the main tool that we have used until now to formalise informal FD languages. The main results of comparative studies of FD languages carried out on the basis of our criteria (Section 4.3) will be recalled in Section 5.

4.1 Making languages suitable for comparison

Let us call X_1 the language we want to compare with the others (X_2, …, X_n) which, we assume, are fully and clearly formalised according to Harel and Rumpe's principles (Section 3) and have identical semantic domains. We distinguish the three aforementioned cases.

Case 1: X_1 has no formal semantics: There are two alternatives.

• The first alternative is to define the syntax and semantics for each FD language individually. That is, we define X_1 independently from X_2, …, X_n. This is what we did in Section 3.4 with OFD. This is also what we did in [6] with OFT.

• The second alternative is to make scale economies and define several languages at once. In [8], we observed that most of the FD languages largely share the same goals, the same constructs and, as we understood from the informal definitions, the same (FODA-inspired) semantics. For this reason, we proposed to define not one FD language but a family of related FD languages (Fig. 10).

We defined a parametric abstract syntax, called FFD, in which parameters (Section 4.2) correspond to variations in $\mathcal{L}_{X_1}, \mathcal{L}_{X_2}, \ldots, \mathcal{L}_{X_n}$. This definition follows, but slightly adapts, the principles of Section 3. The semantic domain (PL, see Definition 2) and semantic function (Section 4.2) are common to all FD variants, maximising semantic reusability. With this method, we are confined to handle languages whose only significant variations are in their abstract syntax. For languages with very different semantic choices, for example [21], it is much harder to describe (and justify) the introduction of variation points in the semantics. Then, we should rather follow either the first alternative in Case 1 (if the language is informal), or Case 2 or 3 otherwise.

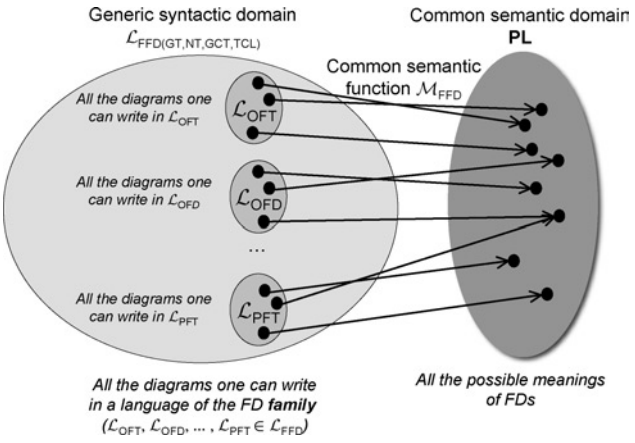


Figure 10 *Meaningful modelling for a family of FD languages with FFD*

Case 2: X_1 has a formal semantics but $\mathcal{L}_{X_1}$, $\mathcal{S}_{X_1}$ and $\mathcal{M}_{X_1}$ need to be clarified: Another frequent case is when a language X_1 actually has a formal semantics, but which does not adhere to Harel and Rumpe's principles. That is, we cannot see explicit and self-contained mathematical definitions of $\mathcal{L}_{X_1}$, $\mathcal{S}_{X_1}$ and $\mathcal{M}_{X_1}$. Typically, $\mathcal{L}_{X_1}$ is clear and self-contained, but $\mathcal{S}_{X_1}$ and $\mathcal{M}_{X_1}$ are not. Most of the time, the semantics of X_1 is given by describing a transformation of X_1's diagrams to another language, say W, which is formal. W does not even need to be an FD language, and usually it is not. Therefore the semantic domain might be very different from the one intuitively thought of for FDs. The main motivation for formalising it in this way is usually because W is supported by tools. The problem is that these kinds of 'indirect', or tool-based, semantics complicate the assessment of the language even more if W is also given a formal semantics in a similarly 'indirect' way, just as X_1.

Proposals of this kind can be found in recent work on FD [18–24]. In this case, in order to facilitate the assessment of the criteria we are interested in, it is convenient to reformulate the way in which the FD languages to be compared are defined. In [17], we have reformulated the FD language proposed by van Deursen and Klint [18] (renamed vDFD) before comparing it with FFD (Section 4.2). If we need to reformulate several semantically similar languages at once, then a generic approach, like in Case 1, could be applied too. The main difference between Cases 2 and 1 is that, in Case 2, formalisation decisions are usually much more straightforward since they have already been made. However, they might be hard to dig out if they are coded in the operational semantics attached to some tool.

Also, formalisations are not necessarily error-free, and errors can thus be discovered when re-formalising [17].

Case 3: X_1 has a formal semantics with clear $\mathcal{L}_{X_1}$, $\mathcal{S}_{X_1}$ and $\mathcal{M}_{X_1}$ but $\mathcal{S}_{X_1} \neq \mathcal{S}_{X_2}, \ldots, \mathcal{S}_{X_n}$: The third and last case is when we have a clear and self-contained mathematical definition of $\mathcal{L}$, $\mathcal{S}$ and $\mathcal{M}$ for all languages (either from the origin or having previously gone through Case 1 or 2) but the semantic domains of the languages to be compared differ, so that they cannot be compared directly for expressiveness, embeddability and succinctness. In this case, we thus need to define a relation between the semantic domains. We met this problem, for instance, when comparing vDFD with FFD [17]. On the one hand, we had $\mathcal{S}_{\text{FFD}} = \text{PL} = \mathcal{PPP}$ (sets of primitive nodes), and on the other, $\mathcal{S}_{\text{vDFD}} = \mathcal{OON}$ (lists of leaf nodes). The latter introduces an order relation on features, and one on products. Comparing languages with different semantic domains is actually possible, but it requires preliminary work which is now explained.

Consider two languages with their syntactic domains $\mathcal{L}_1$ and $\mathcal{L}_2$ and two different semantic domains, respectively, $\mathcal{S}_1$ and $\mathcal{S}_2$. Their semantic functions are $\mathcal{M}_1$ and $\mathcal{M}_2$. We must first compare intuitively the two domains to determine the information they share. We then create a domain $\mathcal{S}$ for this shared information and provide functions $\mathcal{A}_1 : \mathcal{S}_1 \to \mathcal{S}$ and $\mathcal{A}_2 : \mathcal{S}_2 \to \mathcal{S}$, called abstractions. The purpose of these abstraction functions is to remove additional information and keep the 'core' of the semantic domain, where we will perform the comparisons. For example, in [17], we used an abstraction to remove the ordering of features and products from $\mathcal{S}_{\text{vDFD}}$. However, the question of the relevance of this discarded information remains and should be studied carefully.

A simple but frequent case is illustrated in Fig. 11, where domain $\mathcal{S}_1$ contains more information than $\mathcal{S}_2$; we then take $\mathcal{S}_2$ as the common domain. An abstraction $\mathcal{A}$ removes supplementary information from elements of $\mathcal{S}_1$ and maps them in $\mathcal{S}_2$. It then makes sense to look for quasi-translations $\mathcal{T} : \mathcal{L}_1 \to \mathcal{L}_2$ between the languages' syntactic domains. They are translations (Definition 7 in Section 4.3) for the abstracted semantics $\mathcal{A} \circ \mathcal{M}_1$, and can thus be

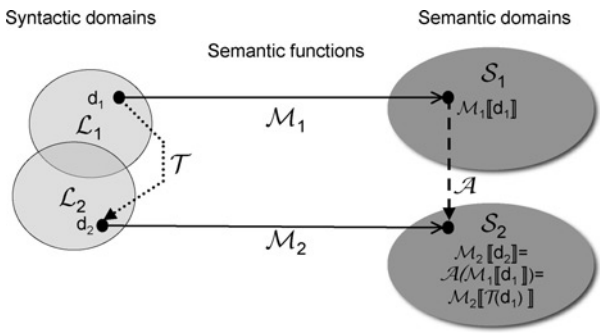


Figure 11 *Abstracting a semantic domain*

used to compare languages for expressiveness, embeddability or succinctness. Hence, if we apply $\mathcal{T}$ to a diagram d_1 in the syntactic domain $\mathcal{L}_1$ we will obtain a diagram d_2 in the syntactic domain $\mathcal{L}_2$ with the same abstracted semantics. Semantically, if we apply the semantic function $\mathcal{M}_1$ to d_1, and then the abstraction function $\mathcal{A}$, we will map to the same element of $\mathcal{S}_2$ as if we apply $\mathcal{T}$ to d_1 and then $\mathcal{M}_2$. That is, $\mathcal{A}(\mathcal{M}_1[\![d_1]\!]) = \mathcal{M}_2[\![\mathcal{T}(d_1)]\!]$.

When applied to more than two languages, this method will create many semantic domains related by abstraction functions. The abstraction functions can be composed and will describe a category [41] of the semantic domains. At the syntactic level, the translations can also be composed to yield expressiveness and succinctness results. Similarly, the composition of embeddings yields an embedding.

4.2 Free feature diagrams

An integral part of our general comparison method is FFD, a parametric abstract syntax that we proposed in [8]. FFD is the main tool that we have used until now to formalise (Case 1) or re-formalise (Case 2) FD languages.

Syntactic domain ($\mathcal{L}_{FFD}$): The definition of the generic syntactic domain of FFD ($\mathcal{L}_{\text{FFD}}$) is a simple generalisation of the syntactic domain of OFT [8], OFD (Section 3.4) and the other surveyed languages (Figs. 2 and 3). FFD stands for 'free feature diagrams' to emphasise its reusability for defining FD languages. In order to cover all the FD languages being formalised, $\mathcal{L}_{\text{FFD}}$ was defined as a parametric construction. Its four parameters (GT, NT, GCT, TCL) correspond to the four variations we have identified in the abstract syntaxes of the languages. That is, we ignored concrete syntax variations such as depicted in Fig. 4.

The abstract syntax variations are as follows.

- The decomposition edges in an FD can form a tree or a DAG. For example, in OFT, they form a tree whereas in OFD, they can form a DAG. Although a DAG is more general than a tree, we should also take trees into account specifically. For this, we introduce the parameter GT: GT (graph type) is either DAG or TREE.

- The type of nodes in an FD may change. In OFT and OFD, the allowed node types (NT) are xor-nodes, and-nodes and opt$_1$-nodes. or-nodes and card-nodes are also included in some other languages, and xor-nodes are not present in others. This variation point corresponds to the parameter NT: NT is a set of boolean functions (operators). It is defined in the same way as in Section 3.4, except that it possibly includes more operators. Noteworthy is the card$_s$[$i..j$] operator introduced to account for EFD [13, 14]. The operator returns TRUE iff at least i and at most j of its s arguments are TRUE. Also, we should note that it is, in principle, possible to add more node types by

doi: 10.1049/iet-sen:20070055

defining their (commutative) boolean operators, in case a new language using a new form of decomposition appears.

- The type of graphical constraints may change. In OFT and OFD, there are no graphical constraints, but most other languages offer this possibility. This variation point corresponds to the parameter GCT:GCT (graphical constraint type) is a set of binary boolean operators. For example, `Requires` (⇒) or `Mutex` (|).

- Finally, the type of textual constraints may change. This variation point corresponds to the parameter TCL (textual constraint language). In general, it can be a subset of the language of boolean formulae where the predicates are the nodes of the FD. In our survey, all the investigated FD languages used CR, (Section 3.4) except one (PFT [12]) which does not have a textual language. Nevertheless, we decided to accommodate for more languages since some authors have proposed to use more powerful languages, for example, boolean logic [22].

The complete formalisation of $\mathcal{L}_{\text{FFD}}$ is available in [11], where it is simply called FFD. Once we had it, it was easy to define all of the surveyed FD languages by simply providing the right parameters. As Table 1 shows, defining an FD language boils down to filling in a row of the table. In order to be complete, the transformation from the concrete FD languages to FFD should also be given. An example is given in Fig. 12 which illustrates the transformation from an EFD (concrete syntax) to an FFD (abstract syntax).

Optional nodes: The last adaptation concerns the optionality. In most FD languages, the nodes can be mandatory or optional, except for EFD [13] where the edges (not the nodes) are mandatory or optional. Both solutions are clearly relevant and therefore for generality, we proposed specific decompositions for optional and mandatory nodes. Let us consider Fig. 13*a*, a very basic FD. OFT and OFD [9] hint that it should be abstracted to *b*, whereas EFD [13] suggests that it should be abstracted to *c*. Because we want to account for both, we take the finer decomposition *d*, adding an opt_1-node f_1? under f_0. f_1? must have f_1 as a son, thus we also add a new edge (that can be omitted in the concrete syntax). We treated mandatory nodes (filled circles) similarly. They can be seen as and_1-nodes.

Semantic domain ($\mathcal{S}_{FFD}$): With FFD, we formalised a set of languages that were inspired from OFT. In particular, we understood that they all shared the same semantic domain: PL (Definition 2). Hence, PL = $\mathcal{S}_{\text{FFD}}$.

Semantic function ($\mathcal{M}_{FFD}$): The genericity of FFD added little complexity to the definition of the semantic function. For example, with respect to the definitions of $\mathcal{S}_{\text{OFT}}$ [4] and $\mathcal{S}_{\text{OFD}}$ (Definitions 3 and 4), the only change is the addition of one more validity rule to account for graphical constraints which are present in some other languages, for example, RFD [10] and EFD [13, 14] (Definition 5).

Definition 5 (valid configuration (in FFD)): A configuration $c \in \mathcal{P}P$ is valid for a $d \in \mathcal{L}_{\text{FFD}}$, noted $c \models d$, iff:

1. c is valid according to Definition 4,

2. c satisfies all graphical constraints: $\forall(n_1, \text{op}_2, n_2,) \in CE$, $\text{op}_2(n_1 \in c, n_2 \in c)$ must be TRUE.

4.3 Comparison criteria

Sections 4.1 and 4.2 have described the method we elaborated to make FD languages amenable to compare with respect to the criteria that we introduced informally at the beginning of Section 4. We now define those criteria more accurately.

Expressiveness: Expressiveness is commonly understood as what can be expressed in a language. For a formal language, we can be more specific: the expressiveness E of a language $\mathcal{L}$ is the part of its semantic domain ($\mathcal{S}$) that it can express, that is, the image of its syntactic domain ($\mathcal{L}$) through its semantic function $\mathcal{M}$. This is what Definition 6 says, and Fig. 14 illustrates. The diagrams in the

Table 1 FD languages defined through FFD

Short name	GT	NT	GCT	TCL
`OFT` [4]	`TREE`	and ∪ xor ∪ {opt_1}	∅	CR
`OFD` [9]	`DAG`	and ∪ xor ∪{opt_1}	∅	CR
`RFD` [10] = `VBFD` [15]	`DAG`	and ∪ xor ∪ or ∪{ opt_1}	{ ⇒ , |}	CR
`EFD` [13, 14]	`DAG`	card ∪{opt_1}	{ ⇒ , |}	CR
`GPFT` [11]	`TREE`	and ∪xor ∪ or ∪ {opt_1}	∅	CR
`PFT` [12]	`TREE`	and ∪ xor ∪ or ∪ {opt_1}	{ ⇒ , | }	∅
`VFD` [7]	`DAG`	card	∅	∅

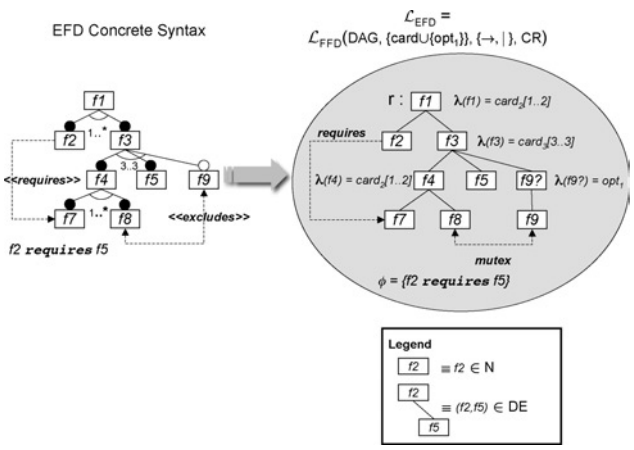


Figure 12 *From EFD (concrete syntax) to FFD (abstract syntax)*

syntactic domain of the language X ($\mathcal{L}_X$) have an image $E(\mathcal{L}_X)$, a subset of X's semantic domain ($\mathcal{S}$).

Definition 6 (Expressiveness): The expressiveness of a language $\mathcal{L}$ is the set $E(\mathcal{L}) = \{\mathcal{M}[\![d]\!] | d \in \mathcal{L}\}$, also noted $\mathcal{M}[\![\mathcal{L}]\!]$. A language $\mathcal{L}_1$ is more expressive than a language $\mathcal{L}_2$ if $E(\mathcal{L}_1) \supset E(\mathcal{L}_2)$. A language $\mathcal{L}$ with semantic domain $\mathcal{S}$ is expressively complete if $E(\mathcal{L}) = \mathcal{S}$.

If $\mathcal{S}$ is the common semantic domain of several languages, say W, X, Y and Z (Fig. 14), their respective expressiveness can be compared.

In the example, we illustrate a situation where, because of their respective definitions, no two languages have the same expressiveness. Also, $\mathcal{L}_Z$ is more expressive than $\mathcal{L}_Y$.

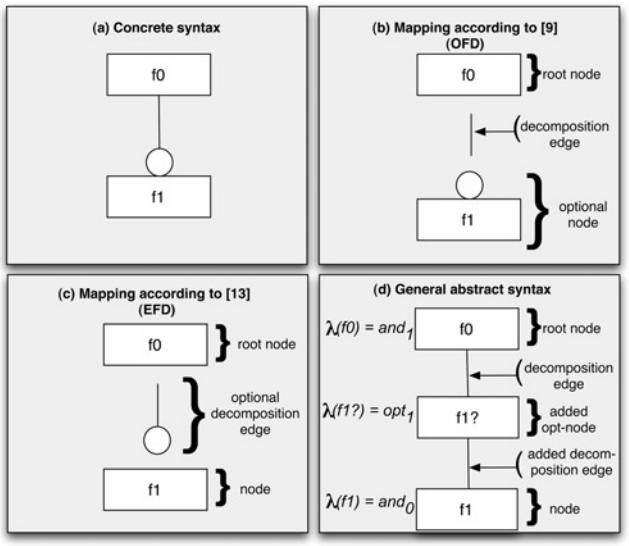


Figure 13 *Three possible abstract syntaxes for optional nodes*

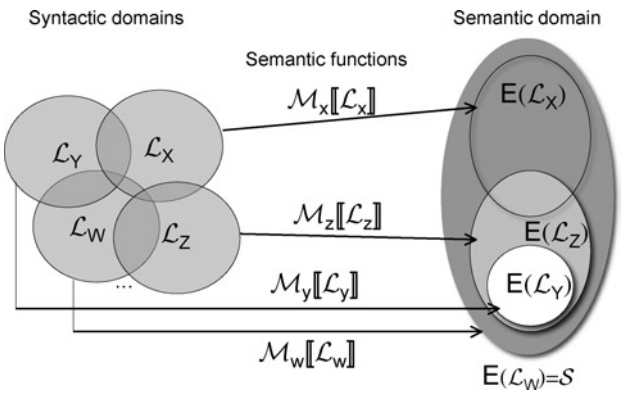


Figure 14 *Comparing expressiveness*

This is written as $E(\mathcal{L}_Z) \supset E(\mathcal{L}_Y)$. The expressiveness of $\mathcal{L}_X$ and $\mathcal{L}_Y$ are disjoint: $E(\mathcal{L}_X) \cap E(\mathcal{L}_Y) = \emptyset$. The expressiveness of $\mathcal{L}_X$ and $\mathcal{L}_Z$ overlap: $E(\mathcal{L}_X) \cap E(\mathcal{L}_Z) \neq \emptyset$. In general, the relationships between the syntactic domains (disjoint, overlapping, equal) of several languages should be considered non-correlated with the relationships existing between their respective semantic domains. This is because the semantic functions can be very different from one language to another.

In Fig. 14, we also notice that $E(\mathcal{L}_W) = \mathcal{S}$. In cases like this, when the image of $\mathcal{L}$ is the whole of $\mathcal{S}$, we say that $\mathcal{L}$ is expressively complete: the part of the semantic domain it can express is the semantic domain itself. Complete expressiveness is a major quality for a language. It ensures that it can express all the intended meanings.

The usual way to prove that a language $\mathcal{L}_2$ is at least as expressive as $\mathcal{L}_1$ is to provide a translation (Definition 7) from $\mathcal{L}_1$ to $\mathcal{L}_2$.

Definition 7 (Translation): A translation is a total function $\mathcal{T} : \mathcal{L}_1 \rightarrow \mathcal{L}_2$ that preserves semantics: $\mathcal{M}_2[\![\mathcal{T}(d_1)]\!] = \mathcal{M}_1[\![d_1]\!]$.

The results that we have obtained studying the expressiveness of FD languages are summarised in Section 5.1.

Since languages compete for expressiveness, it often happens that they reach the same maximal expressiveness (Such as, e.g. $\mathcal{L}_1$ and $\mathcal{L}_2$ in Fig. 15). This is, for instance, the case for programming languages, that are almost all Turing-complete and can thus express the same computable functions. Consequently, we need finer criteria than expressiveness to compare these languages.

The idea is to study the properties of the translations between those languages.

- Do they preserve structure?
- Do they increase size?

doi: 10.1049/iet-sen:20070055

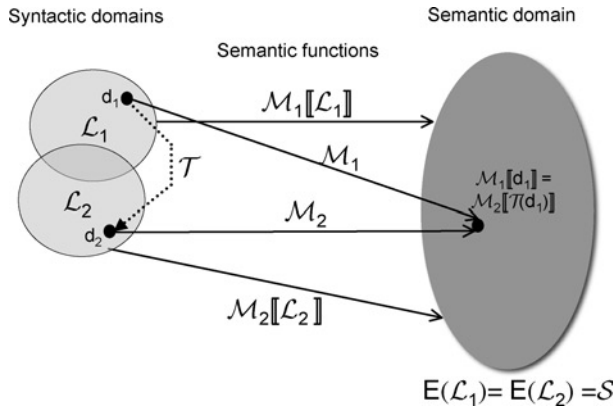


Figure 15 *Translation between expressively complete languages*

The former property is addressed by the concept of embeddability, whereas succinctness takes care of the second.

Embeddability: When two languages have the same expressiveness, in theory, there is a translation between them. However, this translation might destroy the structure of the original diagram.

For textual languages, the requirement to preserve structure (i.e. embeddability) has been called macro-eliminability by Felleisen [42] (inspired by Kleene [43]). Macro-eliminability relies on the assumption that the concerned textual languages have a context-free grammar, which in turn allows to define the translation in a compositional way.

Unfortunately, context-free syntax is not a realistic assumption when dealing with visual languages [27, 28]. In particular, there is no such syntax for DAG-shaped FDs. The compositional definition of the translation can thus not be applied as such. In [7], we have proposed a definition of graphical embeddability which generalises the definition of embeddability for context-free languages. We recall it here (Definitions 8 and 9) in a simplified form.

Definition 8 (Graphical embeddability): A graphical language $\mathcal{L}_1$ is embeddable into $\mathcal{L}_2$ iff there is a graphical embedding (Definition 9) from $\mathcal{L}_1$ to $\mathcal{L}_2$.

Definition 9 (Graphical embedding): A graphical embedding is a translation (Definition 7) $\mathcal{T} : \mathcal{L}_1 \to \mathcal{L}_2$ that is node-controlled [44]: $\mathcal{T}$ is expressed as a set of rules of the form $d_1 \to d_2$, where d_1 is a diagram containing a defined node or edge n, and all possible connections with this node or edge. Its translation d_2 is a subgraph in $\mathcal{L}_2$, plus how the existing relations should be connected to nodes of this new subgraph.

The notion of a node-controlled translation [44] is illustrated in Fig. 16 and further discussed in Section 5.2.

Embeddings are of practical relevance because they ensure that there exists a transformation from one language to the other which preserves the whole shape of the diagrams and generates only a linear increase in size. This way, traceability between the two diagrams is greatly facilitated and tool interoperability is made more transparent. Furthermore, limiting the size of diagrams helps avoiding tractability issues for reasoning algorithms taking the diagrams as an input.

Embeddability can also exist between a language and a subset of itself. A language that is non-trivially self-embeddable [7] is called harmfully redundant (Definition 10). This means that it is unnecessarily complex: all diagrams can be expressed in the simpler sublanguage without loss of structure and with only a linear increase in size.

Definition 10 (Harmful redundancy): A language $\mathcal{L}$ is harmfully redundant iff there is a construct C in $\mathcal{L}$ that has a graphical embedding in $\mathcal{L} \setminus C$.

The embeddability results that we have obtained so far concerning FD languages are summarised in Section 5.2. However, linear translations are not always possible. In this case, the blow-up in the size of the diagram must be measured. This is achieved by examining succinctness.

Succinctness: For languages with same expressiveness, embeddability guarantees that their respective diagrams are (roughly) of the same size (since, by definition, there exists a linear translation between them). When equally expressive languages are not embeddable, succinctness (Definition 11) allows to compare the size of their respective diagrams by computing the size of the diagrams before and after translation from one language to the other. Stated otherwise, succinctness measures the blow-up caused by a change of notation.

Definition 11 (Succinctness): Let $\mathcal{G}$ be a set of functions from $\mathbb{N} \to \mathbb{N}$. A language $\mathcal{L}_1$ is $\mathcal{G}$-as succinct as $\mathcal{L}_2$, noted $\mathcal{L}_2 \preceq \mathcal{G}(\mathcal{L}_1)$, iff there is a translation $\mathcal{T} : \mathcal{L}_1 \to \mathcal{L}_2$ that is within $\mathcal{G}$: $\exists g \in \mathcal{G}, \forall n \in \mathbb{N}, \forall l_1 \in \mathcal{L}_1, |l_1| \leq n \Rightarrow |\mathcal{T}(l_1)| \leq g(n)$. Common values for $\mathcal{G}$ are 'identically' $= \{n\}$, 'thrice' $= \{3n\}$, 'linearly' $= O(n)$, 'cubically' $= O(n^3)$, 'exponentially' $= O(2^n)$. We will omit 'identically'.

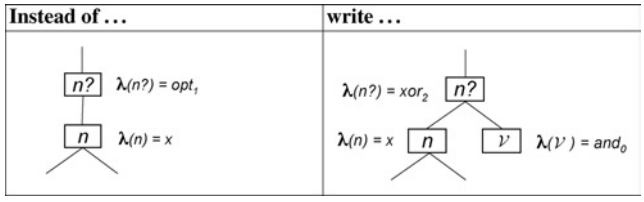


Figure 16 *Node-controlled translation (graphical embedding) of redundant optional node (in OFD concrete syntax)*

If $\mathcal{L}_1$ is more succinct than $\mathcal{L}_2$, this usually entails that $\mathcal{L}_1$'s diagrams are likely to be more readable. Also, if one needs to translate from $\mathcal{L}_1$ to $\mathcal{L}_2$ (e.g., because a tool for achieving some desired functionality is only available in $\mathcal{L}_2$), succinctness will be an indicator of the difficulty to maintain traceability between the original and the generated diagram. Traceability of linear translations is easy but is likely to become more difficult as the size of the generated diagrams grows bigger. However, we should note that this is hard to measure precisely because succinctness does not provide information on the structure of the generated diagrams. (However, looking at the transformation's definition will provide the information.) In this sense, succinctness is a coarser-grained criteria than embeddability. Finally, as we already pointed out concerning embeddability, increases in size are generally not good for the tractability of algorithms. Our current results on succinctness of FD languages are summarised in Section 5.3.

Complexity: When considering decision problems associated with diagrams in a given language, a typical measure of complexity is time complexity, that is, the number of steps that it takes to solve an instance of the problem as a function of the size of the input, using the most efficient algorithm [45]. Exploring the memory usage of the most efficient algorithm is called space complexity. Time and space are usually ranked into complexity classes, like NP-complete, PSPACE-complete... [45].

For example, worst-case execution time of satisfiability checking might grow linearly with the size of the diagrams in some FD languages whereas, in other languages, it may grow exponentially with it. In the latter case, we are likely to face tractability issues.

In our previous work [7], we have studied the complexity of a series of problems for the surveyed FD languages.

- Satisfiability: given a diagram d, is $\mathcal{M}[\![d]\!] \neq \emptyset$ true?
- Equivalence: given two diagrams d_1 and d_2, is $\mathcal{M}[\![d_1]\!] = \mathcal{M}[\![d_2]\!]$ true?
- Model-checking (called product-checking for FDs): given a product c and a diagram d, is $c \in \mathcal{M}[\![d]\!]$ true?
- Intersection: compute a new diagram d_3 such that $\mathcal{M}[\![d_3]\!] = \mathcal{M}[\![d_1]\!] \bigcap \mathcal{M}[\![d_2]\!]$.
- Union: compute a new diagram d_3 such that $\mathcal{M}[\![d_3]\!] = \mathcal{M}[\![d_1]\!] \bigcup \mathcal{M}[\![d_2]\!]$.
- Reduced product: compute a new diagram d_3 such that $\mathcal{M}[\![d_3]\!] = \{c_1 \cup c_2 | c_1 \in \mathcal{M}[\![d_1]\!], c_2 \in \mathcal{M}[\![d_2]\!]\}$.

These are classical problems for languages whose semantic domain is a set. Their relevance in the context of SPL requirements engineering is further elaborated in Section 5.4 and in [8]. Complexity results are important because they help evaluate the scalability of the tool support answering those questions for a given language. Formalisation of both the syntax and semantics is a necessary prerequisite to devise precise questions and make sure that they match intuition. Complexity results then give an indication about the worst case, and how to handle it. Heuristics taking into account the most usual cases can be added to the backbone algorithm, to obtain practical efficiency. Finally, we should note that although formality is required, comparing languages w.r.t. complexity does not require that these languages have the same semantic domains.

5 Language evaluation results

Here, we summarise the results obtained when we have applied our general method of comparative semantics to the FD languages. For the languages defined generically with FFD (Section 4.2), the details and proofs can be found in [7]. The treatment of vDFD [18] is found in [17].

5.1 Expressiveness

For expressiveness, the distinction between languages that only admit trees and the ones that allow sharing of features by more than one parent (DAGs or vDFD) turns out to be important. While tree-shaped languages are expressively incomplete without constraints, OFD are already expressively complete without the constraints, and thus a fortiori RFD, EFD, VBFD and VFD. vDFD are 'almost' trees in that only terminal features (i.e. the leaves) can have multiple parents (justifications), but this is sufficient to obtain expressive completeness.

The expressive incompleteness of tree-shaped diagrams without constraints (in particular, OFT [4] cannot express disjunction) partially justifies a posteriori the proposal [10] (RFD) to add the or operator to OFT. But even so, we do not attain expressive completeness: this language is still unable to express $\text{card}_3[2..2]$, the choice of two features among three. This justifies similarly the proposal [13] (EFD) to use the card operators. Both [10] and [13] also propose to allow DAGs: this extension alone, as we have seen, ensures expressive completeness. But we will see below better justifications in terms of embeddability rather than succinctness.

Designing an FD language is thus essential to include more than trees to reach expressive completeness. Trees, however, are easier to understand and manipulate because they have a compositional semantics. vDFD [18] manages to have both advantages.

5.2 Embeddability

As explained in Section 4.3, a construct can be embedded (or macro-eliminated [42]) in another language if we can express it by a fixed schema.

doi: 10.1049/iet-sen:20070055

An optional node n can be translated into an xor_2-node, say $n?$ with two sons: the original node n, and the TRUE node v which is an and_0-node (i.e. with no son). As we see in Fig. 16, all incoming edges from the parents of n are redirected to the new top node ($n?$), and all outgoing edges to its sons start from the node n. This supports our view that optionality is better treated as an operator.

We constructed an embedding from OFD without constraints (called COFD in [7]) to VFD, presented in Table 2. To save space, we use the textual form for the graphs. For instance, a node bearing an xor_m operator is translated into a node bearing a $card_m[1...1]$ operator. In the next section, we will consider how those embeddings increase the size of the graph. Here, we observe that the VFD diagram resulting from the embedding of a COFD diagram has the same size. This result indicates that card-nodes proposed by Riebisch *et al.* [13] (EFD) can embed all the other constructs. We proposed thus to use them systematically inside tools. We slightly differ from EFD that also uses optional edges: these can be modelled by $card_1[0...1]$-nodes and would be harmfully redundant. We proposed VFD to eliminate this slight drawback. Note that this latter suggestion only concerns abstract syntax. In the concrete syntax, it is probably a good idea to keep optional nodes as this would decrease the size and visual complexity of the diagrams.

5.3 Succinctness

In [7], we also discovered translations that are not expressible as macros, because they depend on the number of neighbours. In this case, it is still interesting to compute the increase in size of the graph, as measured by succinctness. RFD and OFD are of similar succinctness, but, when translating VFD or EFD to OFD, we translate a $card_k$-node to an OFD graph of size $O(k^2)$ [7]. A VFD of size $O(k)$ could contain k $card_k$-nodes, giving a cubic translation at the end: COFD $\leq O(\text{VFD}^3)$. This result indicates again that card-nodes are a useful addition, but for different reasons than presented in [13].

5.4 Complexity

For FDs, solving all the standard problems of Section 4.3 turns out to be practically useful.

- Equivalence of two FDs is needed whenever we want to compare two versions of a PL (for instance, after a refactoring). When they are not equivalent, the algorithm can produce a product showing their difference. For FD languages based on DAGs, and that allow non-primitive features, such as OFD, EFD, VFD, this problem is Π_1-complete [7] (just above NP-complete [45]).

- Satisfiability is a fundamental property. It must be checked for the PL but also for the intermediate FDs produced during a staged configuration [20]. For FD languages based on DAGs, this problem is NP-complete.

- Model-checking (here, also called product-checking) verifies whether a given product (made of primitive features) is in the PL of an FD. It is not as trivial as expected, because the selection performed for non-primitive nodes must be reconstructed. This gives an NP-complete problem. When recording this selection, the problem becomes linear again.

- Union is useful when teams validate in parallel the feature combinations that lead to an acceptable product, without feature interference. Their work can be recorded in separate FDs. The union of these FDs will represent the validated products. For FD languages based on DAGs, this problem is solved in linear time, but the resulting FD should probably be simplified for readability.

- Intersection and reduced product are similar.

The complexity results show the role of non-primitive features: on one hand, it is useful to record them to accelerate the checking of products, but they should not become part of the semantics since this would restrict the expressiveness and strongly reduce the possible transformations of diagrams.

Table 2 Embedding COFD into VFD

Instead of ...	write ...
$opt_1(f)$	$card_1[0...1](f)$
$xor_m(f_1, ..., f_m)$	$card_m[1...1](f_1, ..., f_m)$
$and_s(f_1, ..., f_s)$	$card_s[s...s](f_1, ..., f_s)$

6 Limitations

The main limitation of our work is explicit in its scope: the proposed method and its current results concern only formal language properties related to semantics. In order not to over-interpret our conclusions, one should look at this work with a comprehensive view of model quality in mind. For example, with respect to SEQUAL, in order to be accurate and effective, we have deliberately chosen to address only part of the qualities required from a 'good' modelling language: Domain appropriateness is addressed by looking at language expressiveness; Comprehensibility appropriateness is addressed by looking at embeddability and succinctness; and Technical actor interpretation appropriateness is addressed by looking at decidability, complexity as well as embeddability and succinctness.

Furthermore, we are conscious that our criteria cover only part of each of the aforementioned language qualities. Answering a limited number of questions is a deliberate choice to avoid answering none of them with sufficient accuracy. However, this implies that to obtain a complete

picture of the pros and cons of the languages, further investigation is needed. In Section 7, we will discuss several qualities that we have not addressed at all but that, we think, require similar attention. In particular, our approach evacuates concerns related to concrete syntax, although we think this is a very important topic.

In contrast, a more holistic (quality-wise) attempt to compare feature modelling languages is reported in [46]. However, it is specific in the sense that it concerns the usage of FDs in a particular company, for a given kind of project. This leads us to point out that the notion of a 'good' modelling language is only relative to the context of use of the language. The priorities to be put on the expected qualities and criteria are very likely to be different from one company, or project, to another. This could lead us to relativise in some contexts the importance of formality. But we think that, for FDs, formality is very likely to deliver more than it will cost since (1) the languages are relatively simple, (2) formality can be made largely transparent to the users (hidden behind a graphical concrete syntax), (3) the automation possibilities are many [7, 8, 38] and (4) the information that feature models are used to convey is of critical importance for companies and therefore should suffer no ambiguity.

SEQUAL also helps identify another limitation of our contributions: for the moment, we have only looked at language quality, adopting a theoretical approach. A complementary work is to investigate models empirically. In Section 2, we emphasised the difficulty of such an endeavour because of the limited availability of 'real' FDs. Nevertheless, we do not consider it impossible and can certainly learn a lot by observing how practitioners create and use FDs. Although we have focussed on studying theoretical properties of FD languages, we need to recognise that no formal semantics, nor criteria, can ever guarantee by itself that the languages help capture the right information (neither too little nor too much) about the domain being modelled. We have started to address such concerns recently through the development of reasoning tools, and their applications to case studies. Although VFD have turned out to be a powerful and convenient pivot language for back-end reasoning tools, methodological guidance on how to use them in front ends and how to complement them with other notations had to be provided. In [39], we propose an approach that complements features with an explicit description of the domain assumptions, requirements and specifications underlying them, allowing to deliver and implement a finer-grained notion of satisfiability of a feature model. In [47], we identified the need to separate concerns in feature models, namely PL against software-related variability. From the separated but formaly related variability modelling, we managed to remove some of the ambiguity that the mere formalisation of FDs, as provided in the present and in our past papers, cannot remove. Further empirical research will help us continue to investigate the problem of domain appropriateness.

Moreover, even within the clearly confined scope of our research, we face some threats to validity. Our examination of formal language properties was not supported by tools (except for mere text editors). All the formalisation of, and reasoning (comparisons, demonstrations of theorems) on languages were carried out by humans. Therefore we cannot guarantee that human errors, miss- or over-interpretations are completely absent from our results. In addition, we need to draw the reader's attention on the fact that our formalisations were made only by considering the published documents, and without contacting the authors for clarifications, nor testing their tools. Some of our formalisation choices might therefore only be due to the way things were phrased in the surveyed papers, or to an erroneous understanding from our part.

Finally, concerning the results obtained until now by applying the method, we made clear that there are very relevant FD language proposals [20, 21, 23–25] on which we could not yet apply our method due to lack of time. This is a prioritary topic of future work.

7 Future work

Ultimately, our research aims at accelerating the advent of a standard feature modelling language of an overall excellent quality, including (1) unambiguous and appropriate syntax and semantics and (2) efficient and proved correct reference algorithms. To move forward in this direction, much work is still needed, not only by us.

7.1 Validating the results

Having made explicit the semantics of several feature modelling languages, we need to confront them with their proponents and, more generally, the communities of researchers and practitioners working on the subject. Doing so, we will be able to correct possible misinterpretations (oversimplifications, arbitrary choices etc.) we might have made, but also point out issues that were overlooked in informal definitions. We expect especially lively debates on the issue of edge-based against node-based semantics (see discussion on optional/manadatory nodes in Section 4.2) and on the notion of primitive feature/node (Section 3.4).

Also, testing the tools that implement reasoning algorithms supporting the studied languages would also be a way to obtain a clearer understanding of their semantics. As mentioned in Section 6, some preliminary results can be found in [39, 47]. Those are currently being applied within the context of a real e-Government SPL [48].

doi: 10.1049/iet-sen:20070055

7.2 Extending the results

Our method has been applied to most informal FD languages. A generic formalisation of all of them, FFD, was delivered and has helped gather precise results on them. However, a few specific constructs found in some of the surveyed languages still have to be formalised, most notably, layers, generalisation and implementation links in FORM [9] and binding times in [15].

Concerning other formal languages, the comparative semantics of FFD with vDFD (van Deursen and Klint's FDs) [17] was studied. More recently, several formalisation proposals for FDs appeared in the literature. Comparative semantics should be applied to them as well.

1. Batory [22] provides a translation of FDs to both grammars and propositional formulae. His goal is to use off-the-shelf logic-truth maintenance systems and SAT solvers in feature modelling tools. The semantics of grammars is a set of strings, and thus order and repetition are kept in his first semantics. The semantics of propositional formulae is closer to ours but differs in that decomposable features are not eliminated.

2. In [21], Czarnecki *et al.* define a new FD language to account for staged configuration. They introduce feature cardinality (the number of times a feature can be repeated in a product) in addition to the more usual (group) cardinality. Foremost, a new semantic domain is proposed where the full shape of the unordered tree is important, including repetition and decomposable features. The semantics is defined in a four-stage process where FDs are translated in turn into an extended abstract syntax, a context-free grammar and an algebra. In [20], the authors provide an even richer syntax. The semantics of the latter is yet to be defined, but is intended to be similar to [21].

3. Benavides *et al.* [23] propose to use constraint programming to reason on feature models. They extend the FD language of [21] with extra-functional features, attributes and relations between attributes. Subsequently, they describe tool support based on mapping those FDs to constraint satisfaction problems.

4. Wang *et al.* [25] propose a semantics of FDs using ontologies. A semantic web environment is used to model and verify FDs with web ontology language description logic (OWL DL). The RACER reasoner is used to check inconsistencies during configuration. Their semantics slightly differ from ours, since (1) they omit justifications and (2) they did not eliminate auxiliary symbols.

Here, we just gave very sketchy 'first impressions' of the main existing formal definitions. Each of them now needs to be carefully studied according to the proposed method and criteria.

7.3 Applying the results

Two main applications of our current results can be considered.

- One of the main expected outcomes of our work is the development of efficient tool support for FD languages. Several tools with reasoning capabilities already exist [38] but, for most tasks, they have to face tough tractability issues. Our results (on formalisation and complexity, mainly) can help (1) verify the correctness of these algorithms and (2) devise optimised algorithms.

- Our work suggests VFD as the language currently obtaining the best ranking according to the studied criteria. To make VFD usable, we need to provide them with a concrete syntax (One is proposed in Fig. 3 but was not the outcome of a profound reflection.) and tool support.

Preliminary results on such applications were briefly discussed in Section 6, and more extensively in [39, 47].

7.4 Extending the scope

As mentioned repeatedly in the paper, the scope of our research is limited to a restricted number of qualities and criteria. Studying other qualities and criteria is equally important. In particular, issues related to concrete syntax, ignored in the current paper, are complementary to our current investigations. In our survey of FD languages, we could observe that there were also diverging views on this issue. Despite our focus on semantics, we do not underestimate the impact of a good concrete syntax. In the end, this is the only thing most language users will actually see. Evaluation and improvement of concrete syntax is an area of research that possesses an important body of knowledge which is currently being structured [16, 49], and of which FDs could take advantage. An important topic is reducing the visual complexity of real-size models [49].

Finally, an empirical approach to the quality of FD languages could complement and help validate our theoretical results. For example, complexity results, which are typically worst-case results, should be confronted with observations of the kinds (structure, size...) of models that are actually used by practitioners. For instance, if it turns out that most real FDs are trees (instead of DAGs), then our complexity results should not be considered too pessimistically.

8 Conclusions

The bad news confirmed by this paper is that the current research on variability modelling is fragmented. The modelling of variability, and particularly the use of FDs, can be a precious help in mastering the complexity of variability management in the context of SPLE. FDs

allow to represent in a concise way the commonalities and the variabilities of a whole family of products in terms of their features. Unfortunately, existing research in the field is characterised by a growing number of proposals and a lack of accurate comparisons between them. In particular, the formal underpinnings of FDs need more careful attention.

The nocuous consequences of this situation are: (1) the difficulty for practitioners to choose appropriate feature modelling techniques, (2) an increased risk of ambiguity in models and (3) underdeveloped or inefficient tool support for reasoning on FDs.

The good news that this paper delivers is that there are remedies to this situation. The ones that we propose are: (1) a global quality framework (e.g. Krogstie *et al.*'s SEQUAL) to serve as a roadmap for improving the quality of feature modelling techniques; (2) a set of formally defined criteria to assess the semantics-related qualities of FD languages; (3) a systematic method to formalise these languages and make them ready for comparison and efficient tool automation and (4) a first set of results obtained from the application of this systematic method on a substantial part of the feature modelling languages found in the literature.

Although the road ahead is still quite long, we are confident that the community can take profit of our proposal. It could be used, for example, as part of an arsenal to elaborate a standard feature modelling language. This standard would not suffer from ambiguity, and its formal properties (among others) would be well known, allowing to devise proved correct efficient reference algorithms. We also think that the proposed approach is general enough to be applied to cognate areas where existing modelling techniques face similar challenges. Our work on FDs can thus be regarded as a language (re)engineering examplar available for being transposed to other domains.

9 Acknowledgments

We would like to thank David Harel and Bernhard Rumpe for being a source of inspiration in this work and for their encouragements. Our gratitude also goes to Andreas Metzger, David Benavides, Martin Glinz, Juha Savolainen, the anonymous referees and the attendees of the Software Variability Management (SVM) workshop (held in Helsinki in April 2007) for their constructive comments. This work has been financed by Interuniversity Attraction Poles Programme, Belgian State, Belgian Science Policy, the Belgian National Bank and the FNRS.

10 References

[1] CLEMENTS P.C., NORTHROP L.: 'Software product lines: practices and patterns' in 'SEI Series in Software Engineering' (Addison-Wesley, 2001)

[2] FOREMAN J.: 'Product line based software development – significant results, future challenge'. Software Technology Conf., Salt Lake City, UT, USA, 23 April 1996

[3] POHL K., BOCKLE G., VAN DER LINDEN F.: 'Software product line engineering: foundations, principles and techniques' (Springer, 2005)

[4] KANG K., COHEN S., HESS J., NOVAK W., PETERSON S.: 'Feature-oriented domain analysis (FODA) feasibility study', Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990

[5] COHEN S., TEKINERDOGAN B., CZARNECKI K.: 'A case study on requirements specification: driver monitor'. Proc. Workshop on Techniques for Exploiting Commonality through Variability Management at the Second Int. Conf. Software Product Lines (SPLC'02), 2002

[6] BONTEMPS Y., HEYMANS P., SCHOBBENS P.-Y., TRIGAUX J.-C.: 'Semantics of FODA feature diagrams'. Proc. Workshop on Software Variability Management for Product Derivation: Towards Tool Support, Boston, August 2004, pp. 48–58

[7] SCHOBBENS P.-Y., HEYMANS P., TRIGAUX J.-C., BONTEMPS Y.: 'Generic semantics of feature diagrams', *Comput. Netw.*, 2007, pp. 456–479

[8] SCHOBBENS P.-Y., HEYMANS P., TRIGAUX J.-C., BONTEMPS Y.: 'Feature diagrams: a survey and a formal semantics'. Proc. 14th IEEE Int. Requirements Engineering Conf. (RE'06), Minneapolis, Minnesota, USA, September 2006, pp. 139–148

[9] KANG K.C., KIM S., LEE J., KIM K., SHIN E., HUH M.: 'FORM: a feature-oriented reuse method with domain-specific reference architectures', *Ann. Softw. Eng.*, 1998, **5**, pp. 143–168

[10] GRISS M., FAVARO J., D'ALESSANDRO M.: 'Integrating feature modeling with the RSEB'. Proc. 5th Int. Conf. Software Reuse (ICSR'98), Vancouver, BC, Canada, June 1998, pp. 76–85

[11] EISENECKER U.W., CZARNECKI K.: 'Generative programming: methods, tools, and applications' (Addison-Wesley, 2000)

[12] ERIKSSON M., BÖRSTLER J., BORG K.: 'The PLUSS approach – domain modeling with features, use cases and use case realizations'. Proc. 9th Int. Conf. Software Product Lines (SPLC'05), 2005, pp. 33–44

[13] RIEBISCH M., BÖLLERT K., STREITFERDT D., PHILIPPOW I.: 'Extending feature diagrams with UML multiplicities'. Proc. 6th Conf. Integrated Design and Process Technology (IDPT '02), Pasadena, CA, June 2002

doi: 10.1049/iet-sen:20070055

[14] RIEBISCH M.: 'Towards a more precise definition of feature models'. Position Paper in Modelling Variability for Object-oriented Product Lines, 2003

[15] VAN GURP J., BOSCH J., SVAHNBERG M.: 'On the notion of variability in software product lines'. Proc. Working IEEE/IFIP Conf. Software Architecture (WICSA'01), 2001

[16] MOODY D.L.: 'What makes a good diagram? Improving the cognitive effectiveness of diagrams in IS development'. Proc. 15th Int. Conf. Information Systems Development (ISD'06), 2006

[17] TRIGAUX J.-C., HEYMANS P., SCHOBBENS P.-Y., CLASSEN A.: 'Comparative semantics of feature diagrams: FFD vs vDFD'. Proc. Workshop on Comparative Evaluation in Requirements Engineering (CERE'06), Minneapolis, Minnesota, USA, September 2006

[18] VAN DEURSEN A., KLINT P.: 'Domain-specific language design requires feature descriptions', *J. Comput. Inf. Technol.*, 2002, **10**, (1), pp. 1–17

[19] MANNION M.: 'Using first-order logic for product line model validation'. Proc. 2nd Softw. Product Line Conf. (SPLC'02), San Diego, CA, 2002, LNCS, vol. 2379, Springer, pp. 176–187

[20] CZARNECKI K., HELSEN S., EISENECKER U.: 'Staged configuration using feature models', *Softw. Process Improv. Pract.*, 2005, **10**, (2), pp. 143–169

[21] CZARNECKI K., HELSEN S., EISENECKER U.: 'Formalizing cardinality-based feature models and their specialization', *Softw. Process Improv. Pract.*, 2005, **10**, (1), pp. 7–29

[22] BATORY D.S.: 'Feature models, grammars, and propositional formulas'. Proc. 9th Int. Conf. Software Product Lines (SPLC'05), 2005, pp. 7–20

[23] BENAVIDES D., RUIZ-CORTÉS A., TRINIDAD P.: 'Automated reasoning on feature models'. Proc. 17th Int. Conf. (CAiSE'05) LNCS, Advanced Information Systems Engineering, 2005, vol. 3520, pp. 491–503

[24] SUN J., ZHANG H., LI Y.F., WANG H.: 'Formal semantics and verification for feature modeling'. Proc. 10th IEEE Int. Conf. Engineering of Complex Computer Systems (ICECCS '05), 2005, pp. 303–312

[25] WANG H., FANG L.Y., SUN J., ZHANG H., PAN J.Z.: 'A semantic web approach to feature modeling and verification'. Proc. Int. Workshop on Semantic Web Enabled Software Engineering (SWESE'05), 2005

[26] ASIKAINEN T., MANNISTO T., SOININEN T.: 'A unified conceptual foundation for feature modelling'. Proc. 10th Int. Software Product Line Conf., 2006, pp. 31–40

[27] HAREL D., RUMPE B.: 'Meaningful modeling: what's the semantics of 'semantics'?', *IEEE Comput.*, 2004, **37**, (10), pp. 64–72

[28] HAREL D., RUMPE B.: 'Modeling languages: syntax, semantics and all that stuff, part I: the basic stuff', Technical Report MCS00-16, Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, 2000

[29] KROGSTIE J.: 'Using a semiotic framework to evaluate UML for the development of models of high quality'. Unified modeling language: system analysis, design and develoment issues' (IDEA Group Publishing, 2001), pp. 89–106

[30] KROGSTIE J., SINDRE G., JØRGENSEN H.: 'Process models representing knowledge for action: a revised quality framework', *Eur. J. Inf. Syst*, 2006, **15**, (1), pp. 91–102

[31] HEYMANS P., SCHOBBENS P.-Y., TRIGAUX J.-C., MATULEVIČIUS R., CLASSEN A., BONTEMPS Y.: 'Towards the comparative evaluation of feature diagram languages'. Software and Services Variability Management Workshop Concepts, Models and Tools (SVM07), 2007

[32] LINDLAND O.I., SINDRE G., SØLVBERG A.: 'Understanding quality in conceptual modeling', *IEEE Softw.*, 1994, **11**, (2), pp. 42–49

[33] MOODY D.L.: 'Metrics for evaluating the quality of entity relationship models'. Proc. 17th Int. Conf. Conceptual Modeling (ER '98), Singapore, November 1998, in Ling T.W., Ram S., Lee M.-L. (Eds.): Lecture Notes in Computer Science, vol. 1507, Springer, pp. 211–225

[34] MOODY D.L., SHANKS G.G.: 'Improving the quality of data models: empirical validation of a quality management framework', *Inf. Syst*, 2003, **28**, (6), pp. 619–650

[35] BECKER J., ROSEMANN M., VON UTHMANN C.: 'Guidelines of business process modeling'. Business Process Management, 2000, pp. 30–49

[36] METZGER A., HEYMANS P.: 'Comparing feature diagram examples found in the research literature', Technical Report, University of Duisburg-Essen, 2007

[37] OMG: 'UML 2.0 Superstructure Specification', available at: http://www.omg.org/cgi-bin/doc?formal/05-07-04, Last Checked: 12/08

[38] BENAVIDES D., RUIZ-CORTÉS A., TRINIDAD P., SEGURA S.: 'A survey on the automated analyses of feature models'. Jornadas de Ingeniería del Software y Bases de Datos (JISBD'06), 2006

[39] CLASSEN A., HEYMANS P., SCHOBBENS P.-Y.: 'What's in a feature: A requirements engineering perspective'. Proc. 11th Int. Conf. Fundamental Approaches to Software Engineering (FASE'08), Held as Part of the Joint European Conf. Theory and Practice of Software (ETAPS'08), Budapest, Hungary, March–April, 2008, vol. 4961, pp. 16–30

[40] TARSKI A.: 'Logics, semantics and metamathematics' (Clarendon Press, 1956)

[41] ADAMEK J., HERRLICH H., STRECKER G.: 'Abstract and concrete categories' (Wiley, 1990)

[42] FELLEISEN M.: 'On the expressive power of programming languages'. Proc. 3rd European Symp. Programming (ESOP '90), (1990, edited by Jones N.D. (ed.), vol. 432, LNCS, (Springer), pp. 134–151

[43] KLEENE S.C.: 'Introduction to metamathematics, vol. 1 of bibliotheca mathematica' (North-Holland, Amsterdam, 1952)

[44] JANSSENS D., ROZENBERG G.: 'On the structure of node label controlled graph languages', *Inf. Sci.*, 1980, **20**, pp. 191–244

[45] PAPADIMITRIOU C.H.: 'Computational complexity' (Addison-Wesley, 1994)

[46] DJEBBI O., SALINESI C.: 'Criteria for comparing requirements variability modeling notations for product lines'. Workshop on Comparative Evaluation in Requirements Engineering (CERE'06), 2006, pp. 20–35

[47] METZGER A., HEYMANS P., POHL K., SCHOBBENS P.-Y., SAVAL G.: 'Disambiguating the documentation of variability in software product lines: a separation of concerns, formalization and automated analysis'. Proc. 15th IEEE Int. Requirements Engineering Conf. (RE'07), October 2007

[48] DELANNAY G., MENS K., HEYMANS P., SCHOBBENS P.-Y., ZEIPPEN J.-M.: 'PloneGov as an open source product line'. Proc. Workshop on Open Source Software and Product Lines (OSSPL'07), 2007

[49] MOODY D.L.: 'Dealing with 'Map Shock': a systematic approach for managing complexity in requirements modelling'. Proc. 12th Working Conf. Requirements Engineering: Foundation for Software Quality (REFSQ '06), Luxembourg, 2006

doi: 10.1049/iet-sen:20070055